Departamento de Electrónica, Sistemas e Informática

Especialidad en Sistemas Embebidos

Detection of musical notes using a polyphonic pitch tracking embedded system

TRABAJO RECEPCIONAL que para obtener el **GRADO** de **ESPECIALISTA EN SISTEMAS EMBEBIDOS**

Presenta: **IXCHEL DAYANARA MERCADO FONG Y JOSE LUIS NAVARRO HERNANDEZ**

Asesor **JORGE ARTURO PARDIÑAS MIR**

Tlaquepaque, Jalisco. agosto de 2022.

Detection of musical notes using a polyphonic pitch tracking embedded system

Navarro Hernández José Luis
Electronics department
ITESO
Guadalajara, México
Josel.navarro@iteso.mx

Mercado Fong Ixchel Dayanara
Electronics department
ITESO
Guadalajara, México
ixchel.mercado@iteso.mx

Jorge Arturo Pardiñas Mir
Electronics department
ITESO
Guadalajara, México
jpardinas@iteso.mx

***Abstract*— Detection of polyphonic notes using embedded systems is a field that has not been explored. This article describes a polyphonic tracking embedded system that can detect on real time single and multiple musical notes, as well as their time duration. The solution was designed on an embedded system with a Cortex M-7 core as processor. This was achieved with an algorithm using fast Fourier transform iterating on two buffers in coordination with the direct memory access peripheral. The proposed embedded system was able to detect multiple musical notes on real time. Future works could use the results of this design and export them to a real music editing format.**

Keywords—Musical notes, FFT, pitch tracking embedded system.

I. INTRODUCTION

Music score writing is a skill that requires years of training, and it is also a demanding and time consuming process even for an expert. There are some software solutions that facilitate the process of music score writing; however, these current approaches are either too expensive or limited in characteristics. The typical commercial solutions for music score transcription are available for desktop computer systems; nevertheless, they are expensive or only available through an annual subscription. There are other solutions on smaller systems called single board computers (SBCs) [1]; these systems can detect several musical notes at the same time and are cheaper than traditional desktop computers but are more expensive than an embedded system. Other solutions only offer the detection of single musical notes, limiting an essential part of music known as chords [2]. An affordable solution for music score writing could be designed on an embedded system built with a smaller instruction set architecture (IRA). Nevertheless, this architecture has major design constraints when compared with desktops or SBCs, such as a lower memory space and core frequency.

To properly identify musical notes, it is imperative to define them. A note denotes a musical sound, and they represent pitch and duration. Western music represents musical notes with the next names: *Do, Re, Mi, Fa, Sol, La, Ti,* “Fig. 1”. Musical notes can also include within their names “sharp” and “flat” after them to refer to the pitch between the notes. The pitch notation is standardized by the International Organization for Standardization on ISO 16: 1975-Acoustics standard tuning frequency (Standard musical pitch), for instance, *La* in the fourth scale will correspond to a frequency of 440Hz.

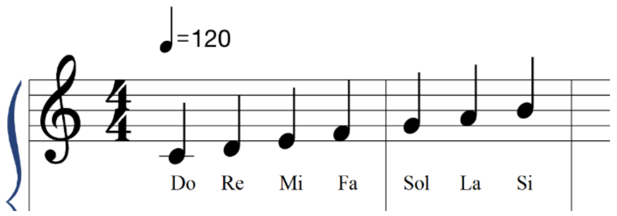


Fig. 1. Musical notes on a score.

The duration of the note is based on the predefined beats per minute (BPM) set by the musician and its time signature. The duration of each note is a fraction of a *whole note*, and they are divided into two parts, and those parts are also divided into two, until a smaller note is obtained, called a *semiquarver or a 16th note* “Fig. 2”. The crochet or quarter note value is one beat; if a time signature indicates 4/4, this means that every bar has a sum of 4 beats and the sum can be obtained by the combination of the different note values.

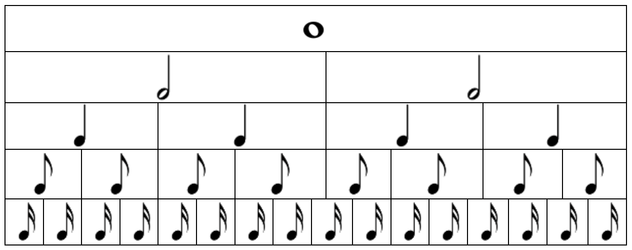


Fig. 2. Types of musical notes.

The objective of this work is to create a polyphonic pitch tracking embedded system that can also determine duration of musical notes that compose a chord by overcoming the design constraints of low memory and core frequency.

This article is structured as follows: Section II describes the design of the algorithm; Section III details the design of the solution on the embedded system to detect musical notes; Section IV shows the obtained results; Section VI describes the conclusions of this work.

II. Algorithm Design

A chord is a harmonic set of multiple pitches/notes played simultaneously. There are many types of chords, but for the development in MATLAB the focus was on major triad chords. A major chord consists of a root note, a major third note and a perfect fifth note; for instance, a major *Do* would have the pitches *Do*, *Mi* and *Sol*. MATLAB was selected as a first tool to design the algorithm to be able to check the results in a short time thanks to its visual aid. To detect a whole chord, the base work was to detect single notes. For this purpose, the sound of a note was sampled and saved in an mp3 file. This file was read with the function "audioread"; the signal was later analyzed with the function spectrogram from MATLAB; this was done to find a pattern that could be recognized.

The spectrogram helped to identify the frequencies that made the sound. This spectrogram indicates which is the fundamental frequency with the highest value in the sample of sound, which was compared to a table of frequencies paired with the name of the notes, as shown in Table 1. The table references the fundamental frequencies of each note.

Table I. Fundamental frequencies of musical notes on the 4th scale

Name of the note	**Hertz**	**Name of the note**	**Hertz**
Do4	261.626	*Sol4*	391.994
Do#4	277.183	*Sol#4*	415.305
Re4	293.665	*La4*	440
Re#4	311.127	*La#4*	466.164
Mi4	329.995	*Ti*	493.883
Fa4	349.228		
Fa#4	369.994		

If the magnitude identified with the spectrogram was in a range of a note in the table with a margin of error of +/- 2Hz, the sound was paired to the note, and therefore, identified. Once the initial idea was validated, the Fast Fourier Transform (FFT) was to have more flexibility configuring the parameters of the process. With this function, it was easier to identify more than one frequency; to identify the chord, the three frequency components with the highest magnitudes were saved in a three-element array and checked if the frequency values lay between +/- 5Hz compared to any note of "Table 1". In case they match, they are compared to a table of chords, as shown in "Table 2". If the three values fulfill the description of the major chords table, they will be identified as a valid chord.

Table II. Major chords

Tonic note	Root	Third	Fifth
Do(C)	*Do*	*Mi*	*Sol*
Do#(C#)	*Do#*	*Fa*	*Sol#*
Re(D)	*Re*	*Fa#*	*La*
Re#(D#)	*Re#*	*Sol*	*La#*
Mi(E)	*Mi*	*Sol#*	*Si*
Fa(F)	*Fa*	*La*	*Do*
Fa#(F#)	*Fa#*	*La#*	*Do#*
Sol(G)	*Sol*	*Ti*	*Re*
Sol#(G#)	*Sol#*	*Do*	*Re#*
La(A)	*La*	*Do#*	*Mi*
La#(A#)	*La#*	*Re*	*Fa*
Ti(B)	*Ti*	*Re#*	*Fa#*

III. System Design

A key difference between the algorithm versions of MATLAB and the embedded system is the limited number of resources. The later one has limited memory and a slower core. The development board used is MIMXRT1010-EVK which has the following important characteristics for the design of the system:

- Arm Cortex-M7 core
- Input clock frequency of 500MHz
- 128MB Flash
- 128KB RAM
- Microphone
- Audio codec WM8960
- Serial Audio Interface (SAI)
- Direct Memory Access (DMA)

A. *Data flow*

The usage of the CPU bandwidth and memory had to be optimized through a special flow of data due to limited hardware resources. This flow of data starts with the microphone, which converts the sound waves into an electrical signal; the signal is sampled into a digital signal by the audio codec and transmitted to the microcontroller via the I2S protocol at a rate of 8000 samples per second. The interface that supports the sample's reception is the SAI peripheral. These samples have a resolution of 16 bits and once the data is received by the microcontroller, the DMA handles the transfers to specific memory locations in a special memory scheme that facilitates the processing of information.

B. *Memory addressing scheme*

The memory used for processing the data is sectioned into two different buffers that can accommodate two audio fragments of 512 samples each. The scheme for accommodating the samples is simple, the DMA will fill one of the buffers, and once it is full, the system will trigger the note detection algorithm to iterate on those samples. At the same time, the DMA will stop adding information to that buffer and instead it will switch to the other buffer “Fig. 3”.

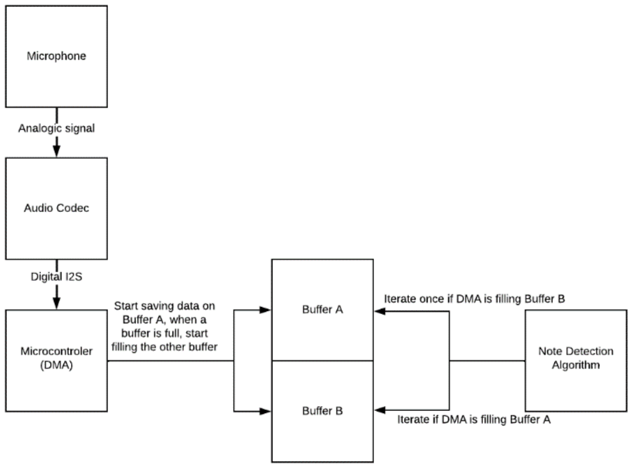

Microphone
Analogic signal
Audio Codec
Digital I2S
Microcontroler (DMA)
Start saving data on Buffer A, when a buffer is full, start filling the other buffer
Buffer A
Buffer B
Iterate once if DMA is filling Buffer B
Iterate if DMA is filling Buffer A
Note Detection Algorithm

Fig. 3. Data flow diagram.

The reasoning behind this memory transferring scheme is that the DMA transfers require no intervention from the core, therefore, while the DMA is storing the data in one buffer, the note detection algorithm can be iterated over the buffer that was previously filled. This technique ensures that the note detection algorithm will deliver results on real time, and that the time needed to process the data is shorter than the time needed to capture the data, which would cause a race condition.

C. *Operating system design*

The implementation of a real time operating system (RTOS) serves two purposes. The first is to ensure the coordination between the task that produces the data (buffer memory filling) and the consumer task (note detection algorithm). The second advantage is that a defined RTOS facilitates additions of new features in the future.

The implemented design consists mainly of three tasks:

- ConsoleTask: This is the first task that gets executed and is used to give the user configuration options.
- StartTransferTask: This task manages the buffer to which the DMA will be transferring the input data every time a buffer gets filled.
- FindNotes: This task iterates the note detection algorithm over the buffer that is full. This algorithm is a port from the MATLAB version.

Since the RTOS has a consumer task and a producing task, the system had to be designed with this in mind and a resource management scheme had to be used. An event-based scheme was chosen because of its simplicity. For its proper functioning, two events on the same event group were required, one for notifying when a transfer of data just started, and the other to notify that a transfer finished filling a buffer. These events support that the algorithm iterates over a buffer that was already full exactly when a new transfer over the next buffer starts. The transitions of the tasks can be found in “Fig. 4”.

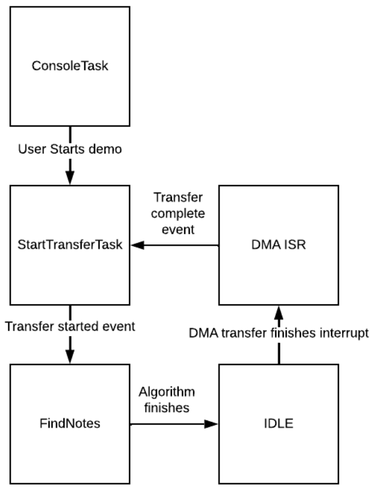


Fig. 4. Tasks transitions.

D. *Note detection in the embedded system*

Just as it was done with the MATLAB algorithm, the base for detecting the notes is the FFT. The precompiled “*arm_math.h”* library was used for this purpose, which is provided by the ISA’s designer. The version of the algorithm used is the complex FFT for float numbers. This means that the data must be converted to the expected format before the transform is performed. After the transformation the algorithm is logic-wise, the same as the MATLAB version.

Because of the system constraints, the algorithm was delimited to only identify notes on two octaves, from *Mi4* to *Re6*. This window was chosen because the frequency delta between two consecutive notes is directly proportional to the pitch. For example, the difference between La3 and La#3 is 13.1 Hz, and the difference between La4 and La#4 is 22.2 Hz. This higher difference is important because the discrete FFT can only detect frequencies on multiples given by the sampling frequency divided by the FFT input size. In the case of our implementation, this frequency is 15.625 Hz; any consecutive notes with a delta higher than this number will have problems being identified, and after the fourth octave, the frequency delta is higher than that multiple.

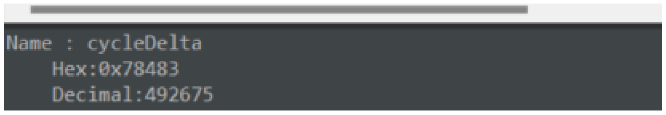
E. *Metronome and time bases*

The user has the option to configure the time base and time signature that will be used to segment the duration of each note. For this purpose, a metronome module was created which can be configured when the demonstration begins with a menu in the console. The calculation of the time bases was performed considering that the shorter period a note lasts on current implementation is of 64 milliseconds. There are three counters for each supported note that can be detected; each time the algorithm is completed, the counter that corresponds to the note detected for each component of the chord adds one. Once a beat has passed, the note counter with the highest value is the one that will be displayed for that beat.

$$NotePeriod = Samples/SamplesPerSecond$$

IV. Results

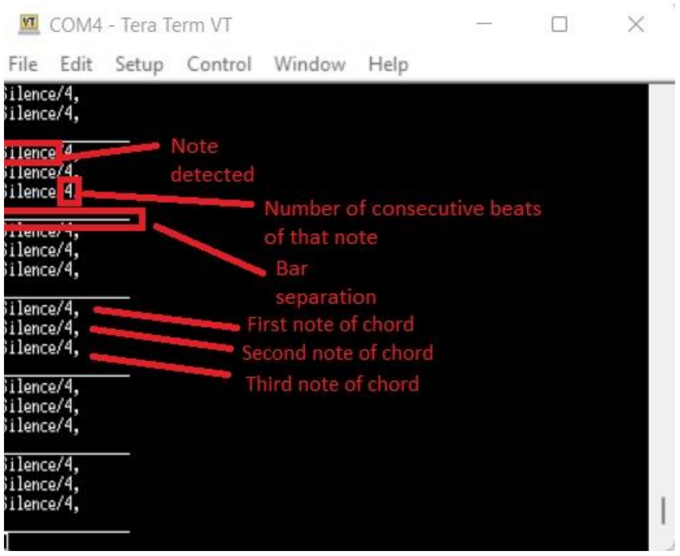
Something important was to ensure that the time of execution of the algorithm took less time than the filling of a 512-sample buffer. “Fig. 5” demonstrates that the deadline is met. The measurement was taken by placing a breakpoint in the beginning of the task and another at the end. The cycle delta measured is the number of clock cycles between two breakpoints, as shown in the image and the number of cycles is 492,675 at 500MHz which corresponds to 0.98 milliseconds.



```
Name : cycleDelta
    Hex:0x78483
    Decimal:492675
```

Fig. 5. Cycle counts of execution.

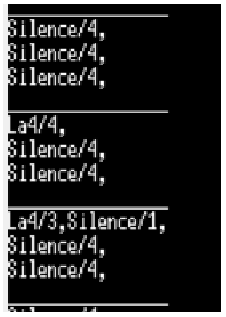
The next “Fig. 6”. shows the output format when there is no sound.



```
ilence/4,
ilence/4,
_________
ilence/4
ilence/4
ilence/4
_________
ilence/4,
ilence/4,
ilence/4,
_________
ilence/4,
ilence/4,
ilence/4,
_________
ilence/4,
ilence/4,
ilence/4,
_________
ilence/4,
ilence/4,
ilence/4,
_________
```

Fig. 6. Description of console format.

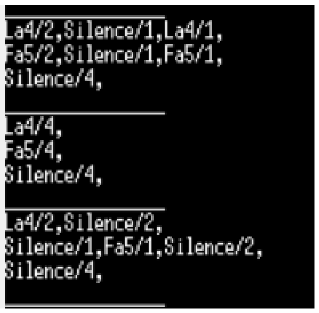
The program was tested with a violin playing a single note. “Fig. 7” shows how the note was played during a full bar and then 3 beats of the next bar.



```
_______________
Silence/4,
Silence/4,
Silence/4,
_______________
La4/4,
Silence/4,
Silence/4,
_______________
La4/3,Silence/1,
Silence/4,
Silence/4,
_______________
```

Fig. 7. Monophonic detection.

“Fig. 8” shows the output when playing 2 simultaneous notes with the violin, in this case, La4 and Fa5.



```
La4/2,Silence/1,La4/1,
Fa5/2,Silence/1,Fa5/1,
Silence/4,
_______________
La4/4,
Fa5/4,
Silence/4,
_______________
La4/2,Silence/2,
Silence/1,Fa5/1,Silence/2,
Silence/4,
_______________
```

Fig. 8. Multiple notes recognition.

V. Conclusions

The polyphonic pitch tracking embedded system presented in this work proves that with the proper design, design constraints can be overcome to correctly identify single and multiple musical notes, as well assigning their proper relative timing value. The timing deadline for the algorithm to finish iterating the recognition algorithm was met.

Identifying 3 different notes played at the same time is a key step to create a solution that can write music scores in the usual format. It is also an important step for chord recognition described in the MATLAB development part of this work.

The current implementation uses the floating-point version of the FFT. Future works could port the program to the 16-bit fixed point version. Using that version would significantly decrease the memory space needs and lower the time needed to run the algorithm. Furthermore, notes could be identified on lower octaves and more time would be given to format results for a proper music score writing program.

[1] R. Schramm and F. Visi, “A polyphonic pitch tracking embedded system for rapid instrument augmentation,” p. 6.

[2] J. K. Patel and E. S. Gopi, “Musical Notes Identification using Digital Signal Processing,” Procedia Computer Science, vol. 57, pp. 876–884, Jan. 2015, doi: 10.1016/j.procs.2015.07.499.